VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Modifikace hry Minecraft pro XNA
Modification of the Game Minecraft for XNA

2013 Tomáš Bureš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Bureš**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Modifikace hry Minecraft pro XNA
Modification of the Game Minecraft for XNA

Zásady pro vypracování:

Cílem práce je vytvořit modifikovanou verzi hry Minecraft v technologii XNA. Hra bude poskytovat většinu běžné funkcionality původní hry Minecraft ve 2D, jako je tvorba krajiny, těžba surovin, pohyb po krajině, "crafiting system", hru více hráčů. Navíc bude hra umožňovat:

1. Rozlévání vody po krajině.
2. Změnu počasí.
3. Rozšířený výběr zbraní pro hru jednoho hráče.

Práce bude obsahovat:
1. Implementaci výše popsané hry.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

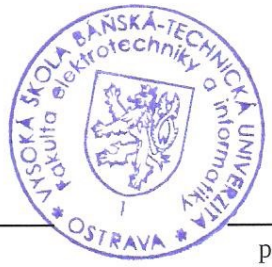
Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] MICROSOFT. XNA Game Studio [online]. [cit. 2012-10-29]. Dostupné z: http://msdn.microsoft.com/en-us/library/cc178930.aspx

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. 4. 2013 .....

Touto cestou bych chtěl poděkovat vedoucímu práce, panu Ing. Davidu Ježkovi, Ph.D. za to, že mi umožnil vypracovat projekt dle vlastního zadání a také za jeho cenné rady a usměrnění projektu do výsledné podoby.

Abstrakt:

Cílem práce je vytvořit modifikovanou verzi hry Minecraft v technologii XNA. Hra bude poskytovat většinu běžné funkcionality původní hry, jako je tvorba krajiny, těžba surovin, pohyb po krajině, "crafting system", vodu, počasí a rozšířený výběr zbraní. Hra bude zpracovaná ve 2D a bude podporovat multiplayer.

Klíčová slova: Počítačová hra, Minecraft, XNA Game Studio, .NET, Multiplayer

Abstract:

The main goal of my dissertation is to create modified version of the game called Minecraft with a usage of XNA technology. Game will provide most of the fuctions of the original game as is landscape creation, harvesting of resources, moving across the land, crafting system, water, weather and extended variety of weapons. Game will be in 2D a will support multiplayer.

Keywords: Computer game, Minecraft, XNA Game Studio, .NET, Multiplayer

Seznam použitých zkratek a symbolů

XNA	-	XNA is Not an Acronym
GUI	-	Graphic User Interface

Obsah

1. Úvod

V dnešní době herního průmyslu se velká část vývojářských studií snaží zaujmout svým produktem co největší množství lidí, aby získaly potřebné finance na následující projekty. Dělají to tím způsobem, že se snaží zaujmout vizuální stránkou hry a velice jednoduchými herními mechanizmy. Jako příklad bych mohl uvést tituly Call of Duty a Battlefield. Pravým opakem je švédská firma Mojang Specifications založena Markusem Perssonem, která se proslavila hrou Minecraft[1][2]. Díky této hře, která vsází na propracované herní mechanizmy a typickou "kostičkovou" grafiku, firma Mojang získala nemalý kapitál a může se dále věnovat projektům, které nebudou mít tak velkou hráčskou základnu, ale zato to bude skupina hráčů, na které se můžou zaměřit a vytvořit hru speciálně pro ně. Příkladem je připravovaná hra the Scrolls. A to je právě ta cesta, kterou bych se chtěl zkusit ve své budoucí kariéře vydat.

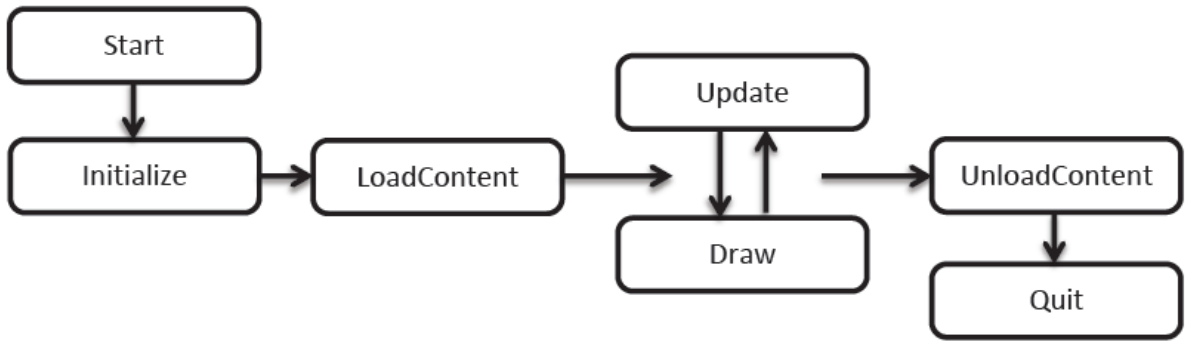
Jako první krok v této cestě jsem si zvolil právě tuto práci. Modifikace hry Minecraft pro XNA. Jako cíl práce jsem si určil vytvoření funkčního prototypu hry. Hra bude disponovat dynamickou interakcí postavy s herním světem a jinými hráči. Tato interakce se bude projevovat hlavně změnou herního světa, úpravy vodních toků a využití střelných či sečných zbraní k nemilosrdné popravě spoluhráče, který se rozhodl znetvořit vaše několik hodin budované dílo.

V samotné práci nejprve rozeberu herní studio XNA, v kterém je hra naprogramovaná. Následně se přesunu k přehledu hry, kde se pokusím vyzdvihnout všechny funkcionality hry a v neposlední řadě se budu věnovat samotné implementaci. Užitečným zdrojem informací při implementaci pro mě byla stránka StackOverflow[5]. V příloze pak naleznete herní manuál.

2. XNA Game Studio

Pro vývoj mojí hry jsem zvolil vývojovou platformu společnosti Microsoft. Jedná se o XNA Game Studio [3][4]. XNA pro svou funkčnost využívá .NET frameworku a slouží jako nadstavba nad grafickým prostředím DirectX. Tato platforma byla vyvinuta za účelem přiblížit vývoj moderních her širší veřejnosti a ulehčit práci již zkušeným vývojářům. V průběhu práce jsem se bohužel dozvěděl, že Microsoft již nadále nebude tuto platformu rozvíjet a verze 4.0 bude její konečnou. Což mě rozhodně nemotivovalo se učit novým věcem, které mi tato platforma nabízí. Doufám, že Microsoft vyvine nějakou adekvátní náhražku této platformy, protože nebýt XNA, tak bychom dnes pravděpodobně neznali hry jako Terraria nebo Magicka.

Funkce samotného XNA je velice jednoduchá a je zobrazena na obrázku. Celá je založena na jednoduché smyčce `Update` - `Draw`.



Obrázek 1 Schéma XNA

- Start - spuštění hry
- Inicializace - V této části dochází k inicializaci tříd.
- LoadContent - V této části probíhá nahrávání všech textur.
- Update - V této metodě je zpracováván veškerý vstup od uživatele a jsou prováděny různé výpočty potřebné pro vykreslení. Například kolize.
- Draw - V této metodě dochází k vykreslení aktuálního stavu hry.
- UnloadContet - Odstraní se veškerý obsah paměti, který byl nahrán v LoadContent.
- Quit - ukončení hry

Tato smyčka se periodicky opakuje. Tuto periodu lze různě nastavovat. Já jsem pro své potřeby zvolil základní nastavení. Toto nastavení vyvolá metodu `Update` a následně `Draw` 60 krát za sekundu.

U tohoto konceptu není rozhodně vhodné psát všechen kód do jedné velké smyčky. Naopak vybízí k tomu, aby každá třída měla svoje metody `Update` a `Draw` a ty byly volány podle potřeby. Vzniknou tak jednotlivé moduly, které mohou být využívány v různých částech programu nebo dokonce i v úplně jiných projektech. Tento systém výstižně popisuje ukázkový diagram na obrázku 2.

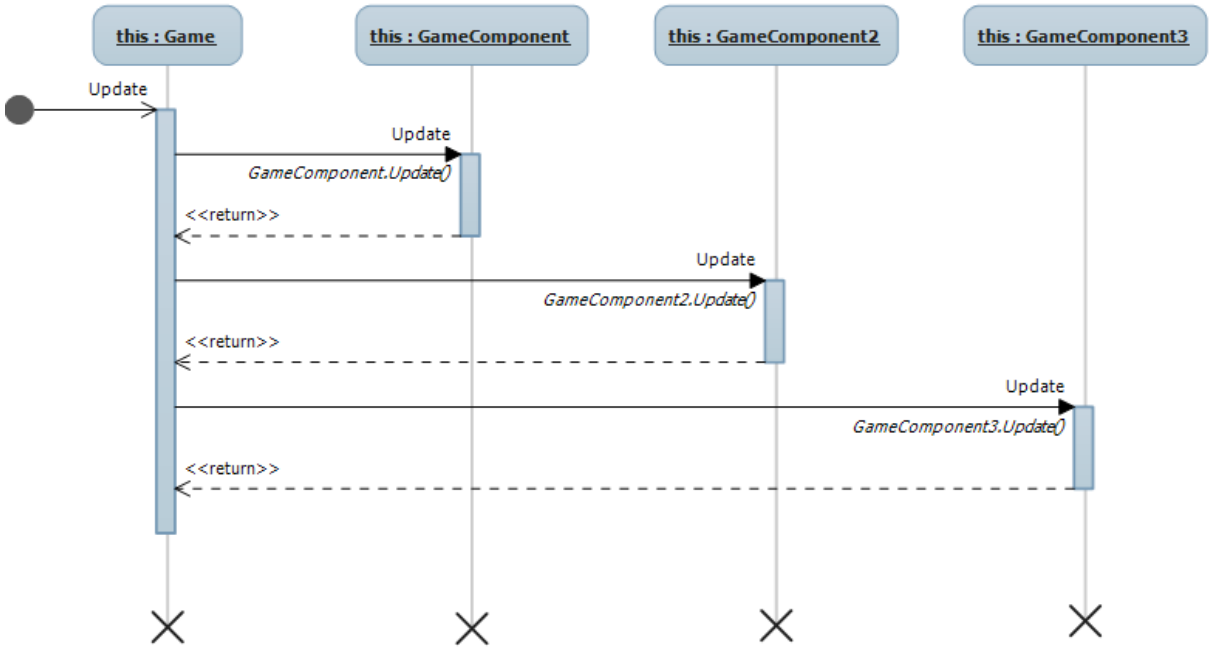
Pro vykreslování mi XNA studio nabízí práci s 2D texturami za pomocí dvou hlavních tříd. Jsou jimi `Texture2D` a `SpriteBatch`. `Texture2D` reprezentuje obrázek (texturu) kterou chceme vykreslit. Pro načtení textur je vhodné využít třetí třídy, tou je `ContentManager`. `ContentManager` je zodpovědný za správu herních dat.

```
Texture2D texture;
ContentManager content;
texture = content.Load<Texture2D>("cesta");
```

Zdrojový kód 1 Práce s texturou

Nyní si potřebujeme definovat objekt, který budeme vykreslovat. K tomuto účelu nám poslouží třída `SpriteBatch` a její tři metody. `Begin`, `Draw`, `End`. Metoda `Begin` připraví všechno potřebné pro vykreslení. Můžeme pomocí parametrů nastavit, jak se bude vykreslování chovat. Následně v metodě `Draw` proběhne samotné vykreslení. Metoda `End` ukončí vykreslování a uvolní obsazené prostředky.

Metoda `Draw` má několik přetížení. Umožní nám vykreslit celou texturu v paměti, případně jenom její výseč. Lokaci, kam chceme vykreslit, obvykle nastavujeme pomocí třídy `Rectangle`. Metoda nám dále umožňuje obrázek transformovat. Využívá k tomu takzvaný `SpriteEffect`. Tento efekt může být žádný, vertikální nebo horizontální. Můžeme upravovat měřítko `Scale` a taky s obrázkem rotovat o zadaný úhel.



Obrázek 2 Sekvenční diagram metody `Update`

3. Přehled hry

Minecraft je sandboxová hra založená na absolutní volnosti. Hráč si z herních kostek různého designu může vytvořit svět dle vlastní představivosti. V mém prototypu hry jsem se snažil tohoto základního herního principu držet a myslím si, že se mi to z velké části povedlo. V následující části uvedu, co se mi do hry podařilo implementovat a co bych v budoucnu ještě implementoval.

3.1. Implementované části

Pohyb, skákání, padání

V herním světě je postavičce umožněno se pohybovat podle nám známým fyzikálním zákonům.

Těžba a pokládání bloků

Za pomocí vhodně vybraného nástroje můžeme vybranou kostku vytěžit a pak umístit na námi vhodně vybrané místo.

Systém počasí (déšť, slunce)

Ve světě je implementované počasí. Může začít pršet. Déšť způsobí rozmočení zavlažení půdy a následné hromadění vody ve světě, zatím co slunce nám vodu a půdu po nějakém čase začne vysoušet.

Dynamický život bloků (růst stromů, trávy)

V případě, že mají kostky hlíny dostatečné množství vody a sluneční energie, vyroste na nich tráva, případně předají část své energie do sazeniček stromů, které po čase také vyrostou.

Crafting

Jedná se o systém, který je dle mého názoru nejklíčovější v celé hře. Umožňuje vyrábět různé věci podle předem daných předpisů.

Střelné zbraně

Možnost výroby a využití střelných zbraní. Každá zbraň má jedinečné vlastnosti (kadence, způsobené poškození). Každá postavička má také takzvané zásahové zóny. Zásah do hlavy způsobí větší zranění, než zásah do děla, případně do ruky.

Smrt

V případě, že postavičce dojdou body zdraví, následuje smrt. Hráč následně ztratí veškerý obsah svého inventáře a je mu nabídnuta možnost respawnu (vzkříšení).

Voda

Voda se nechová podle reálné předlohy z našeho světa. Nicméně jsem se snažil k této předloze co nejvíce přiblížit. Výsledné chování vody je dle mého názoru pro prototyp dostačující.

Hra více hráčů

Do hry se mi taktéž úspěšně podařila implementace hry více hráčů.

3.2. Neimplementované části

Brnění

Příslušný typ brnění postavičce přidá další fiktivní body života, takže vydrží více zásahů.

Animace použití nástrojů a meče

Ve hře jsou tyto prvky implementovány, nicméně jim chybí vizuální stránka.

Pec

Pec bude umožňovat vypalovat železo z železné rudy.

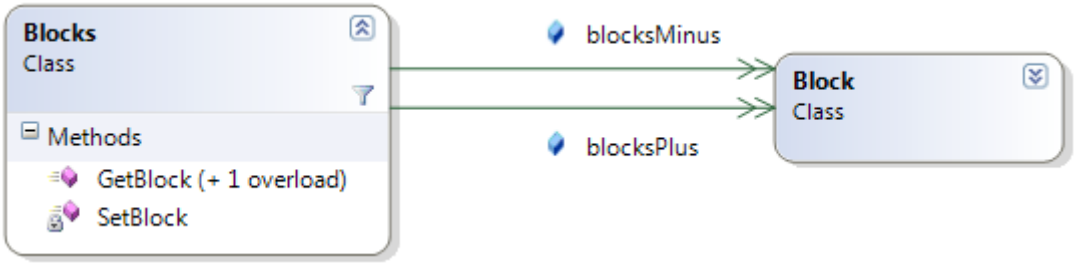
Interakce postavičky s vodou

Jelikož se mi ještě nepodařilo plně synchronizovat vodu ve hře více hráčů, tak jsem do hry zatím neimplementoval žádnou interakci vody a herních postav. Interakce vody a herního prostředí je implementována.

4. Implementace

4.1. Herní plocha

Herní plocha je složená ze základního prvku a tím je objekt třídy `Block`. Tyto bloky jsou uloženy v instanci třídy `Blocks`. Tato třída obsahuje dvě pole. Jedno pro plusové a jedno pro minusové hodnoty herní plochy.



Obrázek 3 Třída `Blocks`

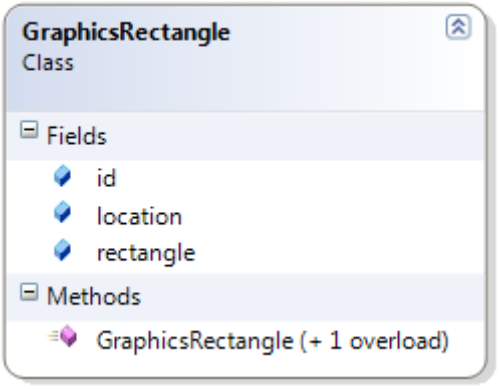
Díky tomuto rozdělení lze nyní se světem pracovat dynamicky a rozšiřovat ho v horizontálním směru o libovolný počet bloků. Toto rozdělení také umožňuje přistupovat do polí pomocí indexů. A to kladných i záporných. K blokům jako k jednomu celku se přistupuje pomocí metod:

```
void SetBlock(int x, int y, Block block)
{
    if (x < 0)
        blocksMinus[Math.Abs(x), y] = block;
    else blocksPlus[x, y] = block;
}
Block GetBlock(int x, int y)
{
    if (x < 0)
        return blocksMinus[Math.Abs(x), y];
    else return blocksPlus[x, y];
}
```

Zdrojový kód 2 SetBlock, GetBlock

4.1.1. Vykreslení herní plochy

K vykreslování herní plochy se využívá třída `Graphics`. Tato třída obsahuje dvě hlavní metody. `Update` a `Draw`. Tyto metody jsou volány příslušnými metodami XNA framworku. Pomocná třída `GraphicsRectangle` slouží pro zjednodušení práce. Atributy této třídy jsou `rectangle`, který určuje pozici a velikost čtverce na obrazovce, lokace bloku ve třídě Blocks a ID samotného bloku.



Obrázek 4 Třída `GraphicsRectangle`

Ve třídě `Graphics` je při inicializaci vytvořeno dvourozměrné pole třídy `GraphicsRectangle`. Toto pole reprezentuje celou herní obrazovku.

```
GraphicsRectangle[,] graphicsRectangle;
```

Zdrojový kód 3 Deklarace pole

V metodě `Update` se následně toto pole naplní daty, která se využijí a za jejich pomoci se vykreslí herní plocha v metodě `Draw`. V neposlední řadě je potřeba atribut `movement`. Ten nám říká, na jakých souřadnicích je aktuálně umístěna herní obrazovka. Inicializační data jsou získána podle pozice hráče a aktuálního rozlišení obrazovky následujícím způsobem.

```
public Point movement;
movement.X = player.location.X - centerOfScreen.X;
movement.Y = player.location.Y - centerOfScreen.Y;
```

Zdrojový kód 4 Inicializace atributu `movement`

Metoda `Update`:

```
public void Update()
{
    int blockId;
    Point blockLocation;
    for (int x = 0; x < countOfBlocks.X; x++)
    {
        for (int y = 0; y < countOfBlocks.Y; y++)
        {
            blockLocation = new Point(x + movement.X / blockSize,
                                      y + movement.Y / blockSize);
            blockId = blocks.GetBlockId(blockLocation.X,
                                        blockLocation.Y);
            graphicsRectangle[x,y] = new GraphicsRectangle(
              new Rectangle(x * blockSize - movement.X % blockSize -
              blockSize,y * blockSize - movement.Y % blockSize,
              blockSize, blockSize),blockLocation , blockId);
...
```

Zdrojový kód 5 Metoda `Update` třídy `Graphics`

Metoda `Draw`:

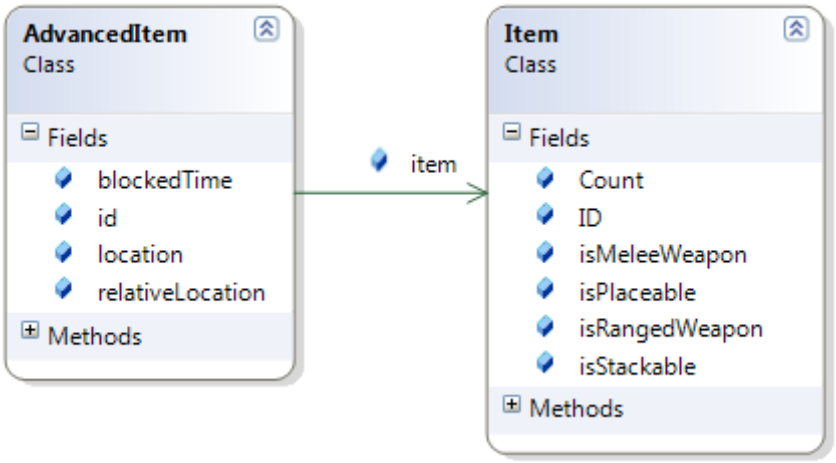
```
public void Draw()
{
    for (int x = 0; x < countOfBlocks.X; x++)
    {
        for (int y = 0; y < countOfBlocks.Y; y++)
        {
            GraphicsRectangle gr = graphicsRectangle[x,y];
            spriteBatch.Draw(terrain, gr.rectangle,
                    SourceRectangle.GetByID(gr.id),Color.White);
        }
    }
}
```

Zdrojový kód 6 Metoda `Draw` třídy `Graphics`

Pro samotné vykreslení se používá třída `SpriteBatch`, což je pracovní plocha XNA frameworku. Metodě `Draw` se následně předá textura, z které se bude vykreslovat, cílová oblast, zdrojová oblast a barva. Statická třída `SourceRectangle` slouží jako zdrojová oblast. Tato oblast se získá voláním metody `GetByID`, jako parametr se předá ID bloku.

4.2. Itemy

Jako první bych rád uvedl, co to item je. Je to vlastně všechno, co si může hráč vzít do inventáře. To znamená, vytěžený blok, zbraň, dřevěná deska a jiné položky herního světa.



Obrázek 5 Třídy `Item` a `AdvancedItem`

Pro práci s itemy se využívají dvě hlavní třídy. Jsou to třídy `Item` a `AdvancedItem`. Ve třídě `Item` jsou uloženy všechny vlastnosti samotného itemu (ID, počet, specifikace a vlastnosti jako jestli jde daný item položit do světa). Ve třídě `AdvancedItem` jsou pak informace důležité pro vykreslení.

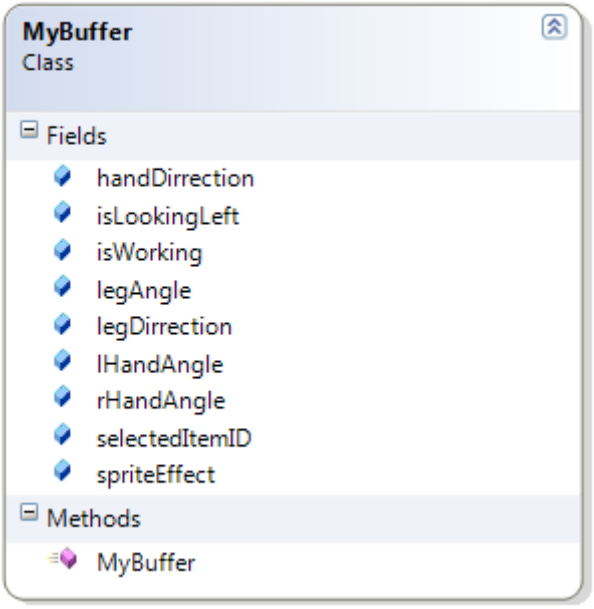
Při práci s itemy jsem narazil na problém. Abych nemusel v každém cyklu posílat přes síť informace o každém itemu, musel jsem do třídy `AdvancedItem` přidat atribut `ID`. Tento atribut slouží jako jedinečné identifikační číslo itemu ve hře. Atribut `blockedTime` se využívá při vyhození itemu z inventáře.

Všechny itemy jsou uloženy v podobě kolekce objektů třídy `AdvancedItem` ve třídě `Items`. Třída `Items` ve své metodě `Add` hlídá výše zmíněné `ID`, aby bylo vždy jedinečné a co možná nejmenší.

Vykreslení pak probíhá ve třídě `Graphics`. V metodě update je aktualizována relativní pozice každého itemu a v metodě `Draw` jsou všechny itemy pomocí metody `DrawItem` vykresleny.

4.3. Postavy

Vykreslování a aktualizace herních postav se provádí taktéž ve třídě `Graphics`. Tato třída dostane přibližně před každým voláním funkce `Update` novou kolekci herních postav. Tato kolekce obsahuje pouze pár informací, jako je aktuální lokace, úhel natočení hlavy, celkové natočení postavy a informaci o tom, jestli se postava pohybuje. Aby se mohla plynule zpracovávat animace postav, musí se vytvořit třída `MyBuffer`. Do této třídy se následně ukládají všechna potřebná data animace.



Obrázek 6 Třída `MyBuffer`

Tyto informace jsou následně uloženy ve slovníku a přiřazeny podle ID postavy.

```
Dictionary<int, MyBuffer> graphicsBuffer;
```

Zdrojový kód 7 Deklarace atributu `graphicsBuffer`

V metodě `Update` se prochází všechny postavy, které jsou uloženy v kolekci `GameObjects`. Přepočítávají se jejich relativní lokaci (lokace na obrazovce) podle atributu `movement` a lokace postavy. Následně se kontroluje, zda je pro danou postavu vytvořený `buffer`, když ne, tak se vytvoří. Poté se volá funkce `Animation`, které se předá jako parametr aktuální postava.

```
public void Update
{
    foreach(GameObject o in gameObjects)
    {
        o.relativeLocation.X = o.location.X - movement.X;
        o.relativeLocation.Y = o.location.Y - movement.Y;
        if (!graphicsBuffer.ContainsKey(o.id))
        {
            graphicsBuffer.Add(o.id, new MyBuffer());
        }
        Animation(o);
    }
}
```

Zdrojový kód 8 Metoda `Update` třídy `Graphics`

V metodě `Animation` se vypočítá úhel nohou a úhel pravé a levé ruky. To vše v závislosti, zda se postava pohybuje, zda drží v ruce nějaký předmět nebo zbraň a všechno tak, aby animace vypadala co nejplynuleji.

Metoda `Draw` která se věnuje vykreslování postav vypadá následovně:

```
public void Draw()
{
    foreach(GameObject o in gameObjects)
    {
        DrawObject(o);
    }
}
```

Zdrojový kód 9 Metoda `Draw` třídy `Graphics`

Metoda `DrawObject` slouží pro postupné vykreslení všech části postavy (hlava, tělo, ruce, nohy) podle dat ve slovníku `graphicsBuffer` a postupně je seřadí, aby vykreslení odpovídalo natočení postavy. To vše pouze v případě, že daná postavička není "mrtvá".

```
private void DrawObject(GameObject o)
{
    if (!o.isDead)
    {
        MyBuffer tmp = graphicsBuffer[o.id];
        if (tmp.spriteEffect == SpriteEffects.None)
        {
```

```
            //postava se vykreslí tak, aby byla natočená vlevo.
        }
        else
        {
             //postava se vykreslí tak, aby byla natočená vpravo.
        }
    }
}
```

Zdrojový kód 10 Metoda `DrawObject` třídy `Graphics`

4.4. Pohyb

Veškerý pohyb se zpracovává v hlavní třídě `MinecraftGame`. Pro rozpoznávání vstupních dat z klávesnice slouží statická třída `Keyboard`, která je součástí XNA frameworku. Voláním metody `GetState` se získá `KeyboardState`, se kterým se nadále pracuje.

```
KeyboardState keyboardState = Keyboard.GetState();
if(keyboardState.IsKeyDown(key))
{
    //zde se zpracovává zmáčknutí a následné držení dané klávesy
}
```

Zdrojový kód 11 Zpracování pohybu

Pro vykonání pohybu se využívají skupiny dvou funkcí. Jedna skupina kontroluje polohu postavy a v případě, že je pohyb umožněn, volá druhou skupinu funkcí, která vykoná samotný pohyb.

První skupina si hlídá polohu postavy. V případě, že se dostane na hranu bloku, zjišťuje se ID bloku, na který se postava chystá vstoupit a pomocí atributu `isMovable` se zjišťuje, zda se tímto blokem dá projít.

```
private bool MoveRight()
{
    if (player.location.X % 64 == 0)
    {
        if (player.location.Y % 64 == 0)
        {
            if (kontroluje dva bloky vpravo)
            {
                Right(); //volá funkci pro samotný pohyb
                return true; //vrací true, protože byl umožněn pohyb
            }
            else return false; //vrací false, pohyb nebyl umožněn
        }
        else //v tomto případě postava nestojí na zemi, pravděpodobně
padá nebo skáče, takže se musí kontrolovat už tři bloky na místo dvou
        {
            if (kontroluje tři bloky vpravo)
            {
                .....
```

```
    }
    else{ //situace, kdy není na hraně bloku a nemusí se nic kontrolovat
        Right();
        return true;
    }
}
```

Zdrojový kód 12 Metoda `MoveRight`

Funkce pro samotný pohyb se chová následovně. V prvé řadě nastaví atribut `isMoving` na hodnotu `true`. Touto změnou informuje třídu `Graphics` o tom, že se postavička pohybuje a dává prostor k animaci. Postavička má určitý prostor k pohybu, aniž by se rolovala celá obrazovka. V případě, že se ale postava dostane moc blízko, začne se spolu s postavou pohybovat celá obrazovka. Pohyb se vykonává tak, že se mění atributy `location` a `movement`. Jak je vysvětleno v následujícím kódu.

```
private void Right()
{
    player.isMoving = true;
    if (player.relativeLocation.X < graphics.centerOfScreen.X + 64) //v
tomhle případě relativní lokace, cože je lokace na obrazovce, nedosáhla
fiktivní hranice, takže se pohybuje pouze s lokací postavy, relativní
lokace je přepočítána před vykreslením.
    {
        player.location.X += krok;
    }
    else //v tomhle případě se již postava dostala ke hranici a musí se
proto začít rolováním celé obrazovky. Relativní lokace je opět
přepočítávána  automaticky.
    {
        player.location.X += krok;
        graphics.movement.X += krok;
    }
}
```

Zdrojový kód 13 Metoda `Right`

Funkce pro vodorovný pohyb jsou volány při stlačení příslušné pohybové klávesy. Zatím co funkce pro vertikální pohyb jsou volány z příslušných metod. Jsou to metody `Jump` pro skok a `Fall` pro pád. Pro to, aby se dalo jednoduše určit, v jakém stavu se postava nachází, se vytvořil atribut `characterState`. Do tohoto atributu se ukládají informace o tom, jestli postavička skáče, padá nebo chodí. Následně v závislosti na aktuálním stavu postavy a zmáčknuté klávesy se vyvolává akce. V případě, že je postava ve stavu `Walking` neboli stavu chůze a je zmáčknuta klávesa pro skok, změní se stav na `Jumping` a je periodicky volána funkce `Jump`, tak dlouho, dokud postava opět nespadne na zem a nezmění se její stav opět na stav chůze.

```
if (keyboardState.IsKeyDown(Keys.W))
{
    if (characterState == CharacterState.Walking)
    {
        characterState = CharacterState.Jumping;
```

```
    }
}

if (characterState == CharacterState.Jumping)
{
    Jump();
}
```

Zdrojový kód 14 Ošetření skoku

V případě, že je postava ve stavu chůze, je na ni automaticky volána funkce `Fall`. To znamená, že pokud pod postavou nebude pevná zem, začne automaticky padat.

```
if (characterState == CharacterState.Walking)
{
    Fall();
}
```

Zdrojový kód 15 Ošetření pádu

Metoda `Jump` pracuje s atributem `forceJump`. Tento atribut udává aktuální sílu skoku. S každým voláním metody `Jump` se tento atribut zmenšuje, až se dostává do záporných čísel a začíná se chovat jako metoda `Fall`.

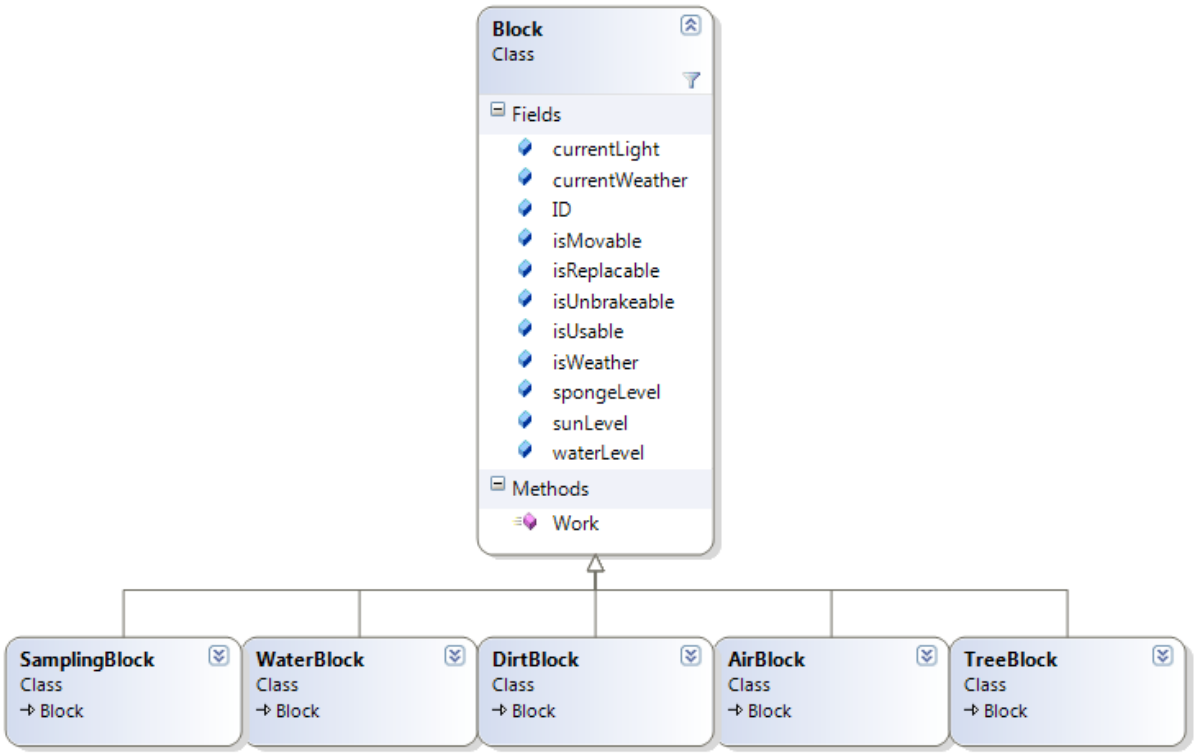
```
private void Jump()
{
    if (forceJump >= 0)
        MultipleDirection(Direction.Up, forceJump);
    else
    {
        bool tmp = MultipleDirection(Direction.Down, -1 * forceJump);
        if (!tmp)
        {
            characterState = CharacterState.Walking;
            forceJump = 17;
        }
    }
    forceJump--;
}
```

Zdrojový kód 16 Metoda `Jump`

Pomocná metoda `MultipleDirection` typu `bool` má dva parametry. Prvním je směr. Ten říká, která funkce pro pohyb se bude volat a druhým parametrem je násobič. Ten udává, kolikrát se funkce zavolá. Metoda následně vrací informaci o tom, jestli pohyb daným směrem proběhl nebo ne. V případě že ne, postava je již na zemi a nastaví se stav postavy na chůzi a taktéž se nastaví počáteční sila skoku na výchozí hodnotu.

4.5. Život bloků

Jak již bylo zmíněno, veškeré bloky jsou uloženy ve třídě `Blocks`. Tato třída obsahuje pole bloků. Třída `Block` obsahuje tyto atributy:



Obrázek 7 Třída `Block`

Objekty třídy `Block` slouží jako reprezentace všech bloků, vyjma těch, pro které je odvozena speciální třída a mají speciální vlastnosti, kterým bude čas věnován později. K simulaci se využívá metoda `Work`, které jsou jako parametry předávány sousedící bloky. Metoda `Work` u třídy `Block` má tři základní vlastnosti. Veškeré vlastnosti budou popsány z pohledu bloku samotného.

- Blok, který je pode mnou, je vždy ve stínu.
- V Bloku, který je pode mnou, neprší.
- Pokud na mě prší, v určitém časovém intervalu generuji do bloku nade mnou vodu.

```
public virtual void Work(Block top, Block bottom, Block left, Block
right)
{
    bottom.currentWeather = Weather.NoWeather;
    bottom.currentLight = Light.Shadow;
    if (counter == weatherCounter)
    {
        if (currentWeather == Weather.Rain)
```

```
            top.waterLevel++;
        counter = 0;
    }
    counter++;
}
```

Zdrojový kód 17 Metoda `Work` třídy `Block`

První speciálním blokem je `AirBlock` a má tyto vlastnosti.

- Blok pode mnou má stejné světelné podmínky jako já.
- Blok pode mnou má po určitém čase vždy stejné počasí jako já.
- V případě, že nesu vodu (`waterLevel` > 0) a blok pode mnou je hlína nebo obecný blok, proměním se na vodní blok (`WaterBlock`).
- V opačném případě předám svojí vodu do bloku pode mnou. Pokud blok pode mnou nemá dost místa, aby pobral všechnu moji vodu, předám mu tolik vody kolik pobere a následně se se zbytkem vody proměním na vodní blok.

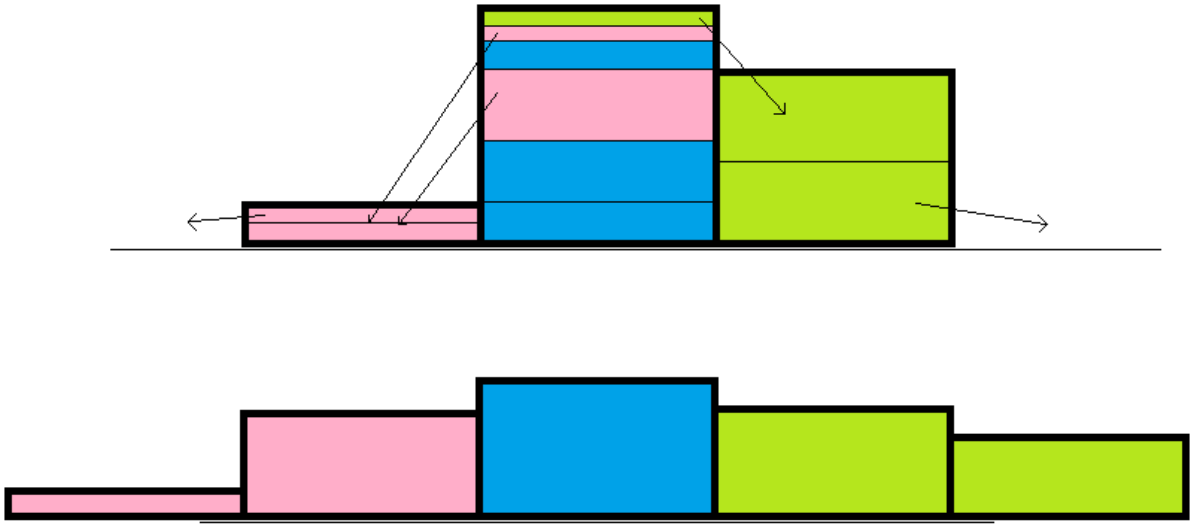
```
if (waterLevel > 0)
{
    if ((IsBlock(bottom) || IsDirt(bottom)))
        return new WaterBlock(waterLevel, currentLight);
    if (IsWater(bottom) || IsSampling(bottom))
    {
        if (bottom.waterLevel == 64)
            return new WaterBlock(waterLevel, currentLight);
        else if (bottom.waterLevel + waterLevel <= 64)
        {
            bottom.waterLevel += waterLevel;
            waterLevel = 0;
        }
        else
        {
            int tmp = bottom.waterLevel + waterLevel - 64;
            bottom.waterLevel = 64;
            return new WaterBlock(tmp, currentLight);
        }
    }
    if (IsAir(bottom))
    {
        bottom.waterLevel += waterLevel;
        waterLevel = 0;
    }
}
```

Zdrojový kód 18 Metoda `Work` třídy `AirBlock`

Dalším speciálním blokem je vodní blok `WaterBlock`.

- Blok, který je pode mnou, je vždy ve stínu.
- V Bloku, který je pode mnou, neprší.

- Vždy, pokud mám možnost, teču dolů.
- Pokud neteču dolů, teču do stran. Pokud mám na obou stranách blok, který má méně vody než já, teču na obě strany a to tak, že v jednom cyklu ztratím přibližně polovinu rozdílu své vody a vody bloku přiléhajícímu.
- v opačném případě teču pouze na jednu stranu.



Obrázek 8 Rozlévání vody

Dalším speciálním blokem je blok hlíny `DirtBlock`.

- Blok, který je pode mnou, je vždy ve stínu.
- V Bloku, který je pode mnou, neprší.
- Pokud na mě prší, saji vodu (dále jako `spongeLevel`).
- Pokud jsem již plně nasáklý a pořád na mě prší, generuji vodu do bloku nad sebou.
- Pokud je nade mnou nebo vedle mě voda a můj `spongeLevel` není na maximální úrovni, tak přilehlou vodu v určité periodě nasávám.
- Pokud na mě svití slunce, přijímám živiny (dále jako `sunLevel`)
- Pokud mám dostatek slunce a vody, přeměním se na trávu
- Pokud na mě přestane svítit slunce a `sunLevel` klesne na nulu, stanu se opět hlínou.
- Část svého `sunLevelu` a `spongeLevelu` předávám do sousedních bloků hlíny.

Posledním speciálním blokem je `SamplingBlocks`. Jeho vlastnosti jsou následující.

- Pokud je jeho úroveň vody nulová, chová se jako `AirBlock`.
- Pokud je jeho úroveň vody nenulová, chová se jako `WaterBlock`.
- Pokud jeho úroveň vody přesáhne určitou hladinu, zaniká a stává se `WaterBlockem`.
- Může být "zasazen" pouze na hlíně. Pokud pod nim není hlína, opět zaniká a stává se `Water` nebo `Air blockem`, podle úrovně vody.
- Z hlíny ve které je zasazen čerpá `sun` a `spongeLevel`.

- Pokud `sun` a `spongeLevel` dosáhne určité úrovně, stává se `TreeBlockem`.

`TreeBlock` nemá žádné speciální vlastnosti. Slouží pouze jako zpětná vazba pro třídu `Blocks`, které když dostane objekt této třídy, vytvoří na jeho místě strom z klasických bloků podle určitého vzorce.

```
private void MakeTree(Point root)
{
    SetBlock(root.X, root.Y, new Block(21));
    SetBlock(root.X, root.Y - 1, new Block(21));
    SetBlock(root.X, root.Y - 2, new Block(21));
    SetBlock(root.X, root.Y - 3, new Block(21));
    SetBlock(root.X - 2, root.Y - 3, new Block(53));
    SetBlock(root.X - 1, root.Y - 3, new Block(53));
    SetBlock(root.X + 1, root.Y - 3, new Block(53));
    SetBlock(root.X + 2, root.Y - 3, new Block(53));
    SetBlock(root.X - 1, root.Y - 4, new Block(53));
    SetBlock(root.X, root.Y - 4, new Block(53));
    SetBlock(root.X + 1, root.Y - 4, new Block(53));
    SetBlock(root.X, root.Y - 5, new Block(53));
}
```

Zdrojový kód 19 Metoda `MakeTree` třídy `Blocks`

Funkce `MakeTree` dostane parametrem lokaci původního `SamplingBlocku` a vytvoří strom. `Block`(21) vytvoří blok dřeva, `Block`(53) slouží pro vytvoření listí, koruny stromu.

Celá simulace života bloků je řízena voláním metody `Update` na instanci třídy `Blocks`. V této metodě je každému bloku `Block` předána informace o jeho sousedních blocích a následně volána metoda `Work`, která celý proces vyhodnotí podle výše sepsaných vlastností a pravidel.

```
public void Update()
{
    for (int y = 0; y < 128; y++)
    {
        for (int x = -199; x < 200; x++)
        {
            top = GetBlock(x, y - 1);
            bottom = GetBlock(x, y + 1);
            left = GetBlock(x - 1, y);
            right = GetBlock(x + 1, y);
            self = GetBlock(x, y).Work(top, bottom, left, right);
            if (self != null)
            {
                if (self.GetType() == typeof(TreeBlock))
                    MakeTree(new Point(x, y));
                else SetBlock(x, y, self);
            }
....
```

Zdrojový kód 20 Metoda `Update` třídy `Blocks`

4.6. Herní menu

XNA studio v základu nedisponuje žádným uživatelským rozhraním. Proto jsem měl dvě možnosti. Někde si nějaké použitelné najít a přizpůsobit si ho nebo si vytvořit svoje. Rozhodl jsem se pro druhou možnost.

Třída `MenuButton` slouží jako základní stavební kámen herního menu. Objekty této třídy reprezentují jednotlivá tlačítka. Do konstruktoru třídy se předávají dva důležité parametry. Text, který má tlačítko vypsat a lokaci, na které se tlačítko vykreslí. Tato třída obsahuje dva důležité atributy.

```
public bool isPressed = false;
private bool isMouseOver = false;
```

Zdrojový kód 21 Atributy třídy `MenuButton`

Atribut `isMouseOver` se využívá uvnitř třídy a v případě, že je kurzor myši nad tlačítkem, nastaví se na hodnotu `true`. To má za příčinu změnu barvy tlačítka.

Atribut `isPressed` se využívá vně třídy. V případě že je na tlačítko kliknuto, nastaví se jeho hodnota na `true`.

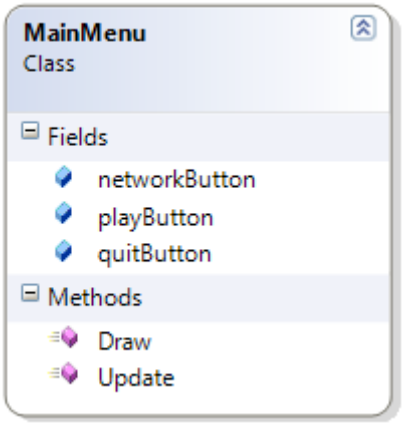
Oba tyto atributy jsou nastavovány z metody `Update`.

```
public void Update()
{
    Rectangle mouseRectangle = new Rectangle(Mouse.GetState().X,
                                        Mouse.GetState().Y, 1, 1);
    if (mouseRectangle.Intersects(location))
    {
        isMouseOver = true;
        mouseStateCurrent = Mouse.GetState();
        if (mouseStateCurrent.LeftButton == ButtonState.Pressed &&
             mouseStatePrevious.LeftButton == ButtonState.Released)
        {
            isPressed = true;
        }
        mouseStatePrevious = mouseStateCurrent;
    }
    else
    {
        isMouseOver = false;
    }
}
```

Zdrojový kód 22 Metoda `Update` třídy `MenuButton`

K nastavení atributu `isMouseOver` se využívá funkce `Intersects`. Tato funkce vrací `true`, pokud se jeden z obdélníků nachází uvnitř druhého. `MouseRectangle` je obdélník, který pronásleduje kurzor.

Dalším krokem je vytvořit třídy, které budou reprezentovat jednotlivé obrazovky herního menu a budou obsahovat všechny potřebné tlačítka a informace o jejich lokaci a zobrazovaném textu.

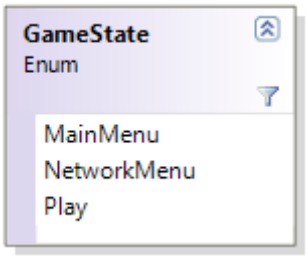


Obrázek 9 Třída `MainMenu`

Kde v každém obdélníku jsou informace o umístění. Následně jsou všechny tlačítka aktualizována a vykreslována v příslušných metodách.

4.7. Herní stavy - neboli GameStates, GUIStates a jiné

Jedná se o hlavní herní stavy, atributy, které určují, v jakém stavu se momentálně hra nachází a řídí tak volání metod `Update` a `Draw` v příslušných třídách.



Obrázek 10 Výčet `GameState`

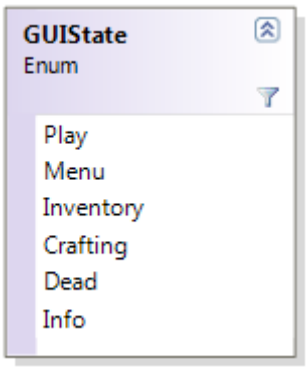
Na následujícím příkladu je popsáno, jak tyto stavy fungují. Po zapnutí hry je `GameState` nastaven na `MainMenu`, to znamená, že ve funkci `Update` a `Draw` se volají příslušné metody pouze na objekt hlavního herního menu. Hlavní menu si v každém cyklu kontroluje, zda náhodou nebylo zmáčknuto nějaké tlačítko. Pokud ano (`isPressed == true`), změní se `GameState` na příslušnou hodnotu, například se zmáčklo tlačítko "Multiplayer", jemuž odpovídá `GameState NetworkMenu`, tak se také nastaví. Také se nastaví atribut `isPressed` u příslušného tlačítka na hodnotu `false`. V následující smyčce `Update Draw` se již nebude vykreslovat hlavní menu, nýbrž menu pro síťovou hru.

```
void Update()
{
    switch (CurrentGameState)
    {
    case GameState.MainMenu:
        if (mainMenu.quitButton.isPressed) this.Exit();
        if (mainMenu.networkButton.isPressed) {
              CurrentGameState = GameState.NetworkMenu;
              mainMenu.networkButton.isPressed = false; }
        if (mainMenu.playButton.isPressed) {
              CurrentConnectionState = ConnectionState.Single;
              CurrentGameState = GameState.Play;
              mainMenu.playButton.isPressed = false; }
        mainMenu.Update();
        break;
    case GameState.NetworkMenu:
        if (networkMenu.serverButton.isPressed) {
              CurrentConnectionState = ConnectionState.Server;
              CurrentGameState = GameState.Play;
              networkMenu.serverButton.isPressed = false; }
        if (networkMenu.clientButton.isPressed) {
              CurrentConnectionState = ConnectionState.Client;
              CurrentGameState = GameState.Play;
              networkMenu.clientButton.isPressed = false; }
        networkMenu.Update();
        break;
    case GameState.Play:
        minecraft.Update(CurrentConnectionState);
        break;
    }
}
```

Zdrojový kód 23 Ověření herního stavu

4.8. Uživatelské rozhraní

Pro zpracování uživatelského rozhraní slouží třída `MyGUIClass`. V této třídě se zpracovává inventář, okno výroby (crafting system) a lišta rychlého výběru. Stejně jako v předchozím případě se zde využívají herní stavy. Fungují na velice podobném principu, takže se zde nebudou podrobně rozepisovat.



Obrázek 11 Výčet `GUIState`

4.8.1. Lišta rychlého výběru

První částí uživatelského rozhraní, které bude věnován čas, bude lišta rychlého výběru. Tato lišta taktéž obsahuje informace o počtu životů. Pro práci s lištou jsou zde vytvořeny tři kolekce.

```
private List<Rectangle> guiSelectorDestinationRectangle;
private List<Rectangle> guiItemsDestinationRectangle;
private List<Rectangle> guiHealthBar;
```

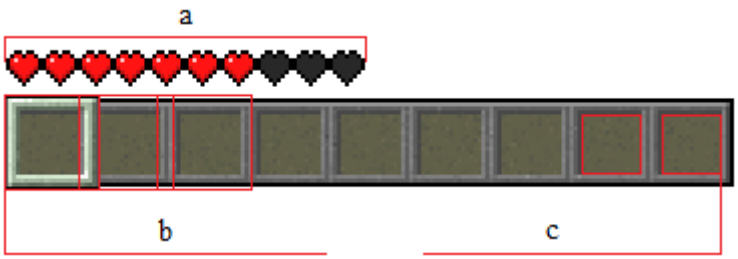
Zdrojový kód 24 Uživatelské rozhraní

Do těchto tří listů jsou v inicializaci uloženy potřebné pozice pro pozdější vykreslení.

```
for (int i = 0; i < 9; i++ )
{
    guiSelectorDestinationRectangle.Add(new Rectangle(left - 2 + i * 40,
                                                top - 2, 48,48));
    guiItemsDestinationRectangle.Add(new Rectangle(left + 6 + i * 40,
                                                top + 6, 32, 32));
}
for (int i = 0; i < 10; i++)
{
    guiHealthBar.Add(new Rectangle(left + 18 * i, top - 30, 18,18));
}
```

Zdrojový kód 25 Inicializace atributů

Vše je detailně vysvětleno v následujícím obrázku.



Obrázek 12 Lišta rychlého výběru a) `guiHelthBar`, b) `guiSelectorDestinationRectangle`, c) `guiItemsDestinationRectangle`

V metodě Update se zpracovává rotace kolečka. `GuiSelectorId` může nabývat hodnot 0 - 8.

```
public void Update()
{
    guiSelectorId = (Mouse.GetState().ScrollWheelValue / 120) % 9;
    if (guiSelectorId < 0) guiSelectorId += 9;
}
```

Zdrojový kód 26 Metoda `Update` třídy `MyGUIClass`

Nyní jsou již všechny hodnoty připraveny a můžou se v metodě `Draw` vykreslit. V liště rychlého výběru se zobrazuje vždy nejspodnější řádek z inventáře postavy.

Při vykreslení se postupuje následovně. V prvé řadě se vykreslí prázdná lišta. Následně se podle `guiSelectorId` vykreslí selektor a to tak, že si z příslušné kolekce vybere obdélník s údaji s příslušným ID. Následuje vykreslení položek z nejspodnější vrstvy inventáře. Pro vykreslení životů slouží funkce `DrawHealth`. Tato funkce nejprve předkreslí prázdné životy (černé srdíčka) a potom přes ně vykreslí aktuální stav životů. Tento stav je uložen v atributu HP.

```
public void Draw()
{
    //vykreslení lišty a selektoru
    spriteBatch.Draw(gui2D, guiDestinationRectangle, guiSourceRectangle,
                     Color.White);
    spriteBatch.Draw(gui2D,
              guiSelectorDestinationRectangle[guiSelectorId],
              guiSelectorSourceRectangle, Color.White);
    for (int x = 0; x < 9; x++)
    {
        if (inventoryItems.inventory[x, 3].ID != 0)
        {
            myDraw.Draw(inventoryItems.inventory[x, 3].ID,
                  guiItemsDestinationRectangle[x]);
        }
    }
    DrawHealth();
}
```

Zdrojový kód 27 Metoda `Draw` třídy `MyGUIClass`

Pro vykreslovaní itemů se vytvořila třída `myDraw`. Tato třída pouze vhodně vybírá texturu, z které se bude vykreslovat podle zadaného ID itemu.

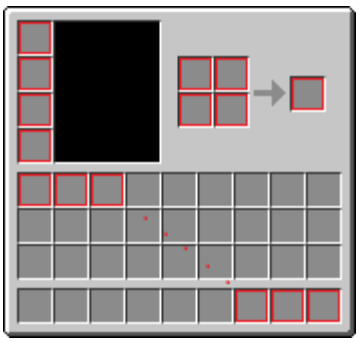
4.8.2. Inventář

V práci s inventářem je využitý podobný systém inicializace jako v předchozí kapitole. Za pomocí dvourozměrného pole třídy `AdvancedRectangle` se vytvoří seznam všech interakčních polí v inventáři. Detail těchto polí je na obrázku 13. Třída `AdvancedRectangle` dále nese informaci o lokaci, díky které se toto pole propojí s datovým polem třídy `Item` (třída `Inventory`).

V metodě `Update` třídy `MyGUIClass` následně vytvořím `mouseRectangle`, což je čtverec velikosti 1 x 1 pixel, který pronásleduje kurzor myšky. Následně procházím dvourozměrné pole (dvourozměrnou kolekci) a zjišťuji pomocí metody `Intersect`, jestli se kurzor nenachází na některém interakčním tlačítku.

```
foreach (List<AdvancedRectangle> list in inventoryRectangle)
{
    foreach (AdvancedRectangle r in list)
    {
        if(mouseRectangle.Intersects(r.rectangle))
        {
            inventoryTarget = r;
            isInventoryTargeted = true;
        }
    }
}
```

Zdrojový kód 28 Metoda Update třídy `MyGUIClass`



Obrázek 13 Inventář

V následující části je zpracovávána interakce uživatele podle následujících pravidel.

Kliknuté levé tlačítko:

- Zvednutí celého stacku (zásobník, hromada)
- Položení celého stacku
- Prohození dvou sacků mezi sebou

- Při kliknutí mimo inventář se stack vyhodí na zem

Kliknuté pravé tlačítko:

- Zvednutí poloviny stacku
- Položení jednoho itemu ze stacku

V inventáři také funguje jednoduchá verze crafting systému. Celá funkčnost tohoto systému je popsána v následující kapitole.

4.8.3. Crafting

Samotná výroba probíhá v podobném okně jako je inventář. Inicializace i pravidla, podle kterých se chová, jsou stejná jako v předchozí kapitole.

Při výrobě se využívá devíti (čtyř) polí. V každém cyklu metody Update se tyto pole prochází, vezmou se `ID` itemů v nich uložených a utvoří se z nich jeden dlouhý řetězec. Tento řetězec se následně porovnává v třídě `CraftDatabase`. Pokud se nalezne shoda, získá se objekt třídy `Item` (`ID` itemu a množství) a provede se výroba.

Výroba krumpáče, zobrazená na obrázku 14, probíhá následovně.

Obrázek 14 Výroba krumpáče

V cyklu získám tyto údaje:

- Kámen - ID 17
- Kámen - ID 17
- Kámen - ID 17

- Nic - ID 0
- Dřevěná tyč - ID 354
- Nic - ID 0
- Nic - ID 0
- Dřevěná tyč - ID 354
- Nic - ID 0

Pomocí funkce `PadLeft` tyto údaje doplním nulami na čtyř místné číslo (řetězec) a spojím je. Výsledný řetězec poté vypadá takto: "00170017001700170000001700170017000017"

S výsledným řetězcem pracuji následovně

```
string craft = "00170017001700170000001700170017000017"
if (craftData.dataBig.ContainsKey(craft))
{
    výsledek který zpracuji = craftData.dataBig[smallCraft];
}
```

Zdrojový kód 29 Craft

4.9. Multiplayer

V dnešní době je dle mého názoru multiplayer jednou z nejdůležitějších částí her. Protože pokud něco vytvoříte a nemáte se s tím komu pochlubit, tak je to vlastně k ničemu. Proto jsem se rozhodl, že se multiplayer pokusím do své hry také implementovat. Jelikož jsem takovou věc implementoval poprvé, rozhodl jsem se, že využiji protokolu TCP/IP. A to z jednoho prostého důvodu, chtěl jsem hlavně řešit, jaké data v jakém množství budu přenášet. Kontroly jestli data dorazila a v jakém pořadí jsem nechal na zmíněném protokolu.

Komunikaci jsem rozdělil na klasické rozpoložení Server - Client. Pro každý prvek jsem si vytvořil příslušnou třídu.

4.9.1. Server

Pokud hra poběží v režimu Server, zavolá se na objekt třídy Server metoda Start. Tato metoda provede všechny potřebné inicializační práce, následně pomocí vlákna zavolá metodu Update, která pracuje v pravidelných intervalech nezávisle na volání metod `Update` XNA frameworku. V této metodě začne probíhat logika serveru a po vyhodnocení všech událostí odešle příslušná data připojeným klientům (zatím obsluhuje jenom lokálního hráče, protože se ještě žádný klient nemohl připojit). Následně se nastaví, na jakém portu bude naslouchat, a opět pomocí vlákna začne čekat na nově příchozí klienty.

```
TcpListener serverSocket = new TcpListener(new IPEndPoint(IPAddress.Any,
                                    8888));

serverSocket.Start();
listenThread = new Thread(Listen);
listenThread.IsBackground = true;
listenThread.Start();
```

```
private void Listen()
{
    while (isListening)
    {
        TcpClient clientSocket = serverSocket.AcceptTcpClient();
        Thread comunicateThread = new Thread(Read);
        comunicateThread.IsBackground = true;
        comunicateThread.Start(clientSocket);
    }
}
```

Zdrojový kód 30 Inicializace třídy `Server`

Po připojení klienta se vytvoří nové komunikační vlákno a předá se mu instance na připojený TcpClient socket. V komunikační metodě `Read` se následně inicializuje spojení. Podle obdrženého ID hráče se zkontroluje, jestli hráč již byl na serveru napojen. V případě že ano, odešle se mu uložený obsah inventáře. Pokud je to hráč nový, odešle se mu prázdný inventář. Následně se hráči odešlou instance tříd `Blocks`, která obsahuje celý svět a instanci třídy `Items`, která obsahuje všechny povalující se itemy ve hře. Nyní je potřeba připojeného hráče předat metodě `Update`, aby mu mohla odesílat změny. To se udělá tak, že se přidá do připravené kolekce `Clients`. Z této kolekce metoda `Update` čerpá při svých aktualizacích. Následuje přechod do čtecí smyčky, která načítá a předává přijatá data ke zpracování.

4.9.2. Client

Při zapnutí hry v roli klienta se opět volá metoda Start. Tato metoda si načte IP adresu a port z přiloženého souboru za pomoci třídy `FileSystem` a pokusí se na tuto IP adresu připojit. V případě úspěchu zpustí komunikační metodu `Read` v novém vlákně.

```
public void Start()
{
    IPEndPoint ip = FileSystem.GetIP();
    if (ip != null)
    {
        try
        {
            clientSocket.Connect(ip);
            Thread readThread = new Thread(Read);
            readThread.IsBackground = true;
            readThread.Start();
        }
        catch
        {
            ConnectionInfo(this);
...
```

Zdrojový kód 31 Inicializace třídy `Client`

Metoda `Read` následně provede inicializační práce, odešle ID a příjme data, která jsou popsána výše, potřebná ke spuštění hry. Provede se spuštění metody `Update`, která běží v cyklu ve vlákně a opět je nezávislá na XNA frameworku. Následně se přejde do čtecí smyčky a jsou načítána příchozí data.

4.9.3. Komunikační data

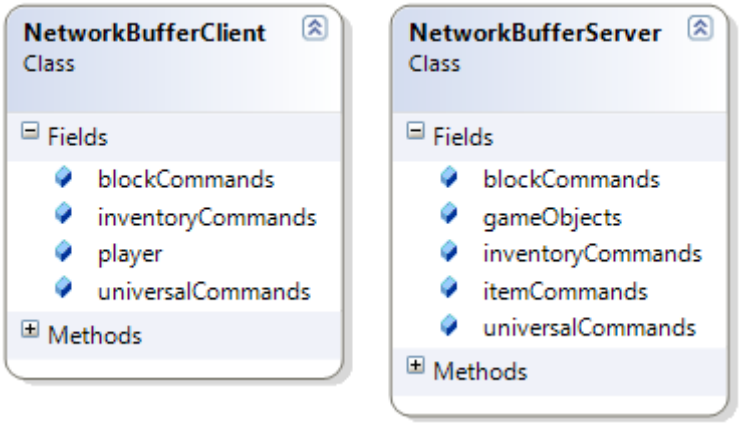
Pro komunikaci mezi serverem a klienty slouží třídy `NetworkBufferServer` a `NetworkBufferClient`. Data z těchto tříd jsou serializována a odesílána v periodických intervalech trvajících 20 milisekund.

4.9.3.1. Příkazy

Komunikační data obsahují seznamy příkazů. Pomocí těchto příkazů jsou přenášena informace o jednorázových změnách. Například změna počasí, změna některého bloku, nově pohozený item a jiné.

Existují čtyři skupiny příkazů.

- Inventory - tato skupina slouží pro práci s inventářem. Například přesun předmětu.
- Block - v této skupině jsou informace o nově vytvořeném nebo zaniklém bloku.
- Item - stejné jako u předchozího případu, akorát s ohledem na item.
- Universal - táto skupina je univerzální. Momentálně slouží pro informace o počasí a efektu střelby.

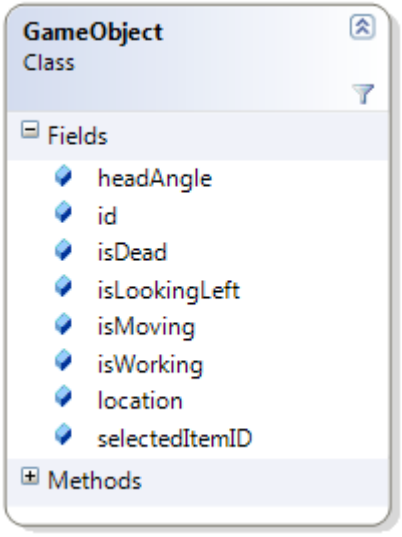


Obrázek 15 Třídy `NetworkBufferClient` a `NetworkBufferServer`

4.9.3.2. Client - server, Server - client

Pro komunikační směr client - server je využívána třída `NetworkBufferClient`. Tato třída kromě příkazů obsahuje informace o klientovi. Respektive o jeho postavě. Tyto informace jsou uloženy ve třídě `GameObject`.

V opačném směru je využívána třída `NetworkBufferServer`. Oproti předchozímu směru navíc přenáší informace o itemech a taky informuje klienta o všech připojených hráčích.



Obrázek 16 Třída `GameObject`

4.9.4. Vlastní serializace

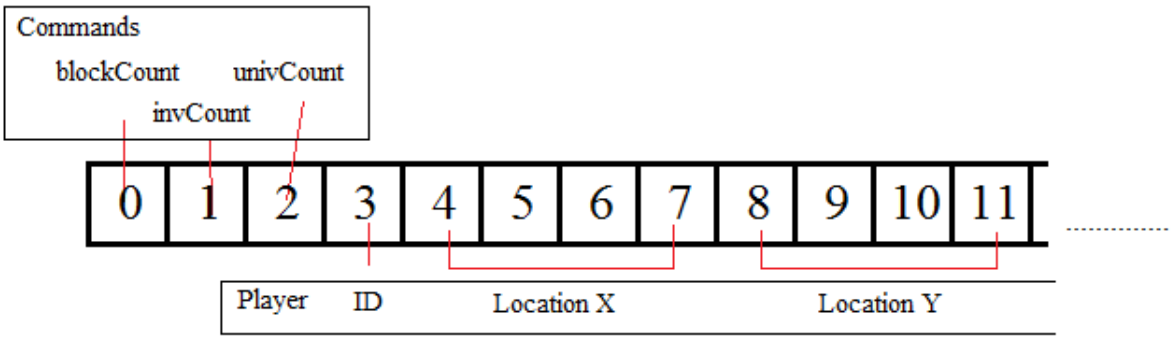
V původní implementované verzi byly celé objekty tříd `NetworkBuffer` přenášeny pomocí binární serializace. Takto serializované objekty ale nadměrně vytěžovaly síťovou linku. Zátěž se pohybovala kolem 70 kb/s na jednoho připojeného klienta. Takové množství dat nebylo akceptovatelné. Proto jsem musel navrhnout vlastní serializaci. Pro tyto účely jsem si vytvořil třídu `MyNetConvertor`. Tato třída má dvě hlavní metody. Metoda `Serialize` dostane parametrem `NetworkBuffer` a vrací pole serializovaných bytů. Druhá metoda `Deserialize` funguje opačně. V následujícím příkladě je popsána serializace třídy `NetworkBufferClient`.

Do prvních pár pozic pole jsou uloženy informace a počtu jednotlivých příkazů, kterou budou odesílány (0 - 2 byte). Následuje vyhrazený prostor pro informace postavě (celkem 16 bytů, 3 - 18).

- ID hráče - 1 byte
- Lokace X - 4 byty
- Lokace Y - 4 byty
- Úhel hlavy - 4 byty
- ID vybraného itemu - 2 byty
- Informace o natočení, pohybu, a jestli postava pracuje, jsou uloženy v posledním bytu

Následují vyhrazené pozice pro příkazy bloků (17 bytů na jeden příkaz), příkazy inventáře (17 bytů na jeden příkaz) a univerzální příkazy (13 bytů na jeden příkaz).

Na takto serializovaná data připadá přibližně 2 kb/s dat na jednoho hráče. Nevýhodou této metody je, že při úpravě posílaných dat, musí přijít také úprava metod serializace a deserializace.



Obrázek 17 Serializace

4.10. Ukládání dat

Ukládání dat je zajištěno pomocí binární serializace a je implementováno pouze pro demonstrační účely hry. Veškeré práce se soubory jsou zajištěny pomocí statické třídy `FileSystem`. Tato třída umí ukládat a načítat bloky, itemy a obsah inventáře. Uložená data, konkrétně pole všech bloků, při použití binární serializace zabírají téměř 20 mb a samotné ukládání a načítání trvá několik sekund. Jelikož dnešní hardware nabízí ohromné množství úložného místa, tak jsem tenhle problém neřešil. Nicméně věřím, že vhodně upravenou serializací, tak jak je použito v předchozí kapitole, by se velikost uložených dat a rychlost ukládání a načítání dala zredukovat na nepatrné velikosti.

5. Závěr

Na závěr bych chtěl objasnit smysl této bakalářské práce. Jelikož jsem veškeré zdroje využil z již licencované a velice úspěšné hry Minecraft, nemyslím si, že by bylo vhodné hru někde zveřejňovat nebo se ji pokoušet distribuovat. Mým cílem bylo vytvořit funkční prototyp hry podle specifikací zadání. Taktéž jsem si dal za cíl vytvořit projekt, na kterém si vyzkouším, co všechno tvorba takové hry obnáší. Získat co nejvíce zkušeností a poučit se z chyb, které jsem v průběhu udělal a následně jsem je musel opravovat, což mě stálo mnoho úsilí a času. Samozřejmě by šel každý aspekt hry vylepšit a naprogramovat lépe, případně celý předělat, ale to bych se potom dostal do začarovaného kruhu, s kterým se vývojář musí potýkat. V průběhu práce jsem měl neustále další a další nápady na inovaci hry, bohužel čas byl omezený, tak jsem se musel striktně omezit na zadání a pokusit se jej vypilovat do co nejlepší podoby. A dle mého názoru se mi takový projekt podařilo vytvořit.

6. Reference

[1] PERSSON, Markus. *Minecraft* [online]. 2011 [cit. 2013-04-29]. Dostupné z: www.minecraft.net

[2] *Minecraft Wiki* [online]. 2011 [cit. 2013-04-29]. Dostupné z: http://www.minecraftwiki.net/wiki/Minecraft_Wiki

[3] MICROSOFT. XNA Game Studio [online]. [cit. 2012-10-29]. Dostupné z: http://msdn.microsoft.com/en-us/library/cc178930.aspx

[4] LAMR, Tomáš. Vývoj her na platformě XNA [online]. 2007 [cit. 2013-04-29]. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Ondřej Krajíček. Dostupné z: <http://theses.cz/id/bkp2yb/>

[5] Stackoverflow, c#. *Stackoverflow* [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://stackoverflow.com/questions/tagged/c%23

Příloha A. Obsah přiloženého DVD

- Text bakalářské práce
- Hra Minecraft 2D a její zdrojový kód

Příloha B. Uživatelský manuál

Pohyb

Pro pohyb jsem zvolil klasickou kombinaci kláves W,S,A,D

W - Skok
S - pomalá chůze
A - Chůze vlevo
D - Chůze vpravo

GUI

I - inventář
Kolečko Nahoru/Dolů - Výběr aktivního pole rychlé volby
Q - vyhození jedné kostky z aktivního pole rychlé volby

Těžba a pokládání kostek

Pro vytěžení kostky je třeba zvolit vhodný nástroj, jinak se nám daná kostka nemusí povést vytěžit. Například když se pokusíme kámen vytěžit jiným předmětem než krumpáčem, tak ho sice po čase rozbijeme, ale nebudeme ho moct zvednout a využít jinde.

Potřebné nástroje pro těžbu:

Dřevo - nic/sekera
Hlína - nic/lopata
Kámen - Krumpáč
Ruda/Uhlí - Krumpáč
"Badrock" - nelže vytěžit - slouží jako hranice světa

Kostku položíme, pokud ji máme vybranou v poli rychlé volby a klikneme do volného prostoru pravým tlačítkem myši.

Pro těžbu musíme vybrat vhodný nástroj a následně držet levé tlačítko myši pro zahájení těžby. Po určitém čase závislém na zvoleném nástroji je kostka vytěžena a je nám umožněno si ji přidat do inventáře. V případě že těžbu přerušíme, potřebný čas se resetuje a těžíme znova.

Střelba

V případě že vlastníte střelnou zbraň, můžete jí využít ke střelbě. Zbraň musí být opět vybrána v poli rychlé volby a samotnou střelbu zahájíte zmáčknutím a držením levého tlačítka myši.

Interakce s předměty ve hře

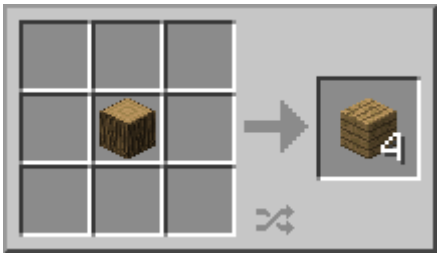
Ve hře se nacházejí kostky, na které když pravým tlačítkem myši klikneme, vyvoláme speciální událost.

Crafting table - Otevře se příslušné okno pro Crafting table a umožní nám vytvářet předměty

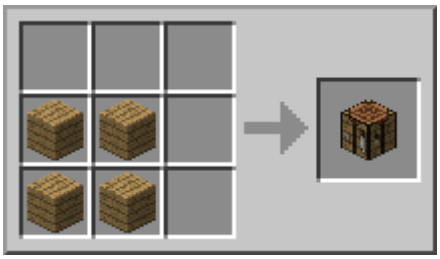
Crafting system

Jak jsem již zmínil, výroba předmětů je klíčovým prvkem hry. Samotná výroba může probíhat v inventáři, kde má hráč možnost využít pracovní plochy 2x2 čtverce, případně pokud si již předem vyrobil Crafting table, může využít právě ji, kde má pracovní plochu 3x3 čtverce.

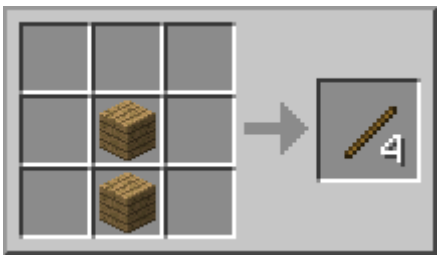
Možnosti výroby shrnu v následujícím přehledu:



Výroba 1 Dřevěné desky



Výroba 2 Crafting table



Výroba 3 Tyčky

Výroba 4 Sekera

Výroba 5 Krumpáč

Výroba 6 Lopata

Výroba 7 Meč